CLAIMS

1. A method of rendering a scene using a graphics processor comprising:

configuring a multithreaded processing unit within the graphics processor to enable processing of samples independent of an order in which the samples are received; and

processing the samples independent of the order in which the samples are received by the multithreaded processing unit to render at least a portion of the scene.

2. The method of claim 1, further comprising:

configuring the multithreaded processing unit to disable processing of samples independent of an order in which the samples are received to minimize visual artifacts in the rendered scene; and

processing a portion of the samples in the order in which the samples are received by the multithreaded processing unit.

3. The method of claim 2, wherein the portion of the samples includes samples within intersecting objects.

4. The method of claim 2, wherein the portion of the samples includes samples within coincident objects.

5. A graphics processor for multithreaded execution of program instructions comprising:

at least one multithreaded processing unit configured to receive samples in a first order to be processed by program instructions associated with at least one thread including:

a scheduler configured to receive the program instructions, determine availability of source data, and schedule the program instructions for execution to process the samples in a second order independent of the first order;

a resource tracking unit configured to track the availability of the source data; and

a dispatcher configured to output the program instructions in the second order to be executed by the at least one multithreaded processing unit.

6. The graphics processor of claim 5, wherein the samples include at least one of vertices, primitives, surfaces, fragments and pixels.

7. The graphics processor of claim 5, further comprising a thread control buffer configured to store program counters, each program counter associated with one of the at least one thread.

8. The graphics processor of claim 5, further comprising an instruction cache configured to store the program instructions.

9. The graphics processor of claim 5, wherein the scheduler is configured to schedule the program instructions for execution to process the samples in a second order where the second order is the same as the first order.

10. The graphics processor of claim 5, wherein the scheduler is configured to schedule the program instructions for execution after ordering the program instructions.

11. The graphics processor of claim 10, wherein the ordering is based on the number of cycles each of the program instructions has been in an instruction window unit.

12. The graphics processor of claim 5, wherein the scheduler is configured to determine availability of computation resources within the at least one multithreaded processing unit.

13. The graphics processor of claim 12, wherein the resource tracking unit is configured to track the availability of the computation resources.

14. The graphics processor of claim 5, wherein the at least one multithreaded processing unit is configured to allocate storage resources to the at least one thread.

15. The graphics processor of claim 5, wherein the at least one multithreaded processing unit is configured to maintain thread state data for the at least one thread.

16. The graphics processor of claim 15, wherein a portion of the thread state data indicates whether the thread is either assigned to a sample or is available to be assigned to a sample.

17. A computing system comprising:

a host processor;

a host memory, the host memory storing programs for the host processor;

a system interface configured to interface with the host processor; and

a graphics processor for multithreaded execution of program instructions including:

at least one multithreaded processing unit configured to receive samples in a first order to be processed by program instructions associated with at least one thread including:

a scheduler configured to receive the program instructions, determine availability of source data, and schedule the program instructions for execution in a second order independent of the first order;

a resource tracking unit configured to track the availability of the source data; and

a dispatcher configured to output the program instructions in the second order to be executed by the at least one multithreaded processing unit.

18. The computing system of claim 17, wherein the host memory is configured to interface with the system interface.

19. The computing system of claim 17, wherein the host memory is configured to directly interface with the host processor.

20. A method of processing a first program instruction associated with a first thread and a second program instruction associated with a second thread comprising:

receiving a first sample to be processed by the first program instruction associated with the first thread before receiving a second sample to be processed by the second program instruction associated with the second thread;

determining that first source data required to process the first program instruction are not available;

determining that second source data required to process the second program instruction are available; and

dispatching the second program instruction to process the second sample in the execution unit prior to dispatching the first program instruction to process the first sample in the execution unit.

21. The method of claim 20, further comprising determining that a position hazard does not exist between a position of the first sample and a position of any other sample being processed by a program instruction in the execution unit.

22. The method of claim 21, further comprising, prior to the determining that a position hazard does not exist, disabling processing of samples independent of an order in which the samples are received using a programmable mode.

23. The method of claim 20, further comprising determining that a position hazard does not exist between a position of the second sample and a position of any other sample being processed by a program instruction in the execution unit.

24. The method of claim 20, further comprising:

retaining as state information the position of the first sample received to be processed by the first program instruction associated with the first thread; and

updating the state information when the first thread has completed execution.

25. The method of claim 20, further comprising:

retaining as state information the position of the second sample received to be processed by the second program instruction associated with the second thread; and

updating the state information when the second thread has completed execution.

26. The method of claim 20, further comprising allocating storage resources to the second thread.

27. The method of claim 20, further comprising allocating storage resources to the first thread.

28. A method of using a function call to configure a graphics processor comprising:

detecting that a multithreaded processing unit within the graphics processor supports processing of samples independent of an order in which the samples are received for at least one sample type; and

issuing a function call to configure the multithreaded processing unit to enable processing of samples independent of an order in which the samples are received for the at least one sample type.

29. The method of claim 28, wherein the sample type is at least one of higher-order surface, vertex, primitive, pixel, and fragment.

30. An application programming interface for a graphics processor comprising a function call to configure a multithreaded processing unit within the graphics processor to enable processing of samples independent of an order in which the samples are received.

31. The application programming interface of claim 30, wherein the function call configures the multithreaded processing unit within the graphics processor to enable processing of the samples independent of the order in which the samples are received for at least one sample type.

32. The application programming interface of claim 31, wherein the sample type is at least one of higher-order surface, vertex, primitive, pixel, and fragment.

33. An application programming interface for a graphics processor comprising a function call to configure a multithreaded processing unit within the graphics processor to disable processing of samples independent of an order in which the samples are received.

34. The application programming interface of claim 33, wherein the function call configures the multithreaded processing unit within the graphics processor to disable processing of the samples independent of the order in which the samples are received for at least one sample type.

35. The application programming interface of claim 34, wherein the at least one sample type is at least one of a vertex sample type and a pixel sample type.